**Ejercicios:**

**1.** Tenemos un sistema que usa claves de 16 bytes. Por razones de seguridad vamos a proteger la clave de tal forma que ninguna persona tenga acceso directamente a la clave. Por ello, vamos a realizar un proceso de disociación de la misma, en el cuál tendremos, una clave fija en código, la cual, sólo el desarrollador tendrá acceso, y otra parte en un fichero de propiedades que rellenará el Key Manager. La clave final se generará por código, realizando un XOR entre la que se encuentra en el properties y en el código.

La clave fija en código es B1EF2ACFE2BAEEFF, mientras que en desarrollo sabemos que la clave final (en memoria) es 91BA13BA21AABB12. ¿Qué valor ha puesto el Key Manager en properties para forzar dicha clave final?

La clave fija, recordemos es B1EF2ACFE2BAEEFF, mientras que en producción sabemos que la parte dinámica que se modifica en los ficheros de propiedades es B98A15BA31AEBB3F. ¿Qué clave será con la que se trabaje en memoria?

**Respuesta 1.**

El valor que ha puesto el Key manager es el siguiente: 20553975c31055ed que hemos obtenido aplicando una operación XOR en hexadecimal ya que es la forma de ambas claves. El valor lo hemos obtenido a través del software Mascript (adjuntamos pantallazo en el archivo de procesos) dividiendo el código de la clave fija B1EF2ACFE2BAEEFF entre la clave final en memoria 91BA13BA21AABB12.

**Respuesta 2.**

La clave con la que se trabajará en memoria será 8653f75d31455c0 que la hemos obtenido con una operación XOR en hexadecimal a través de la aplicación Mascript(adjuntamos pantallazo en el archivo de procesos) dividiendo el código de la clave fija es B1EF2ACFE2BAEEFF entre la de ficheros de propiedades B98A15BA31AEBB3F.

**2.** Dada la clave con etiqueta “cifrado-sim-aes-256” que contiene el keystore. El iv estará compuesto por el hexadecimal correspondiente a ceros binarios (“00”). Se requiere obtener el dato en claro correspondiente al siguiente dato cifrado:

TQ9SOMKc6aFS9SlxhfK9wT18UXpPCd505Xf5J/5nLI7Of/o0QKIWXg3nu1RRz4QWElezdrLAD5LO4USt3aB/i50nvvJbBiG+le1ZhpR84oI=

Para este caso, se ha usado un AES/CBC/PKCS7. Si lo desciframos, ¿qué obtenemos?

¿Qué ocurre si decidimos cambiar el padding a x923 en el descifrado?

¿Cuánto padding se ha añadido en el cifrado?

Se valorará positivamente, obtener el dato de la clave desde el keystore mediante codificación en Python (u otro lenguaje).

**Cifrado-sim.aes-256:**

A2CFF885901A5449E9C448BA5B948A8C4EE377152B3F1ACFA0148FB3A426DB72

**Respuesta 1.**

**Obtenemos el mensaje claro: “**Esto es un cifrado en bloque típico. Recuerda, vas por el buen camino. Ánimo.”

**iv:** 00000000000000000000000000000000

**Mensaje descifrado quitando el padding añadido:**

4573746f20657320756e206369667261646f20656e20626c6f7175652074c3ad7069636f2e2052656375657264612c2076617320706f7220656c206275656e2063616d696e6f2e20c3816e696d6f2e.

**Respuesta 2.**

Mensaje descifrado añadiendo el padding:

4573746f20657320756e206369667261646f20656e20626c6f7175652074c3ad7069636f2e2052656375657264612c2076617320706f7220656c206275656e2063616d696e6f2e20c3816e696d6f2e0100000000000000000000000000000010

Añade 10 bite en hexadecimal

**3.** Se requiere cifrar el texto “KeepCoding te enseña a codificar y a cifrar”. La clave para ello, tiene la etiqueta en el Keystore “cifrado-sim-chacha20-256”. El nonce “9Yccn/f5nJJhAt2S”. El algoritmo que se debe usar es un Chacha20.

¿Cómo podríamos mejorar de forma sencilla el sistema, de tal forma, que no sólo garanticemos la confidencialidad sino, además, la integridad del mismo? Se requiere obtener el dato cifrado, demuestra, tu propuesta por código, así como añadir los datos necesarios para evaluar tu propuesta de mejora.

**cifrado-sim-chacha20-256:** AF9DF30474898787A45605CCB9B936D33B780D03CABC81719D52383480DC3120

**Chacha20**

**texto cifrado:**

nonce en HEX = f5871c9ff7f99c926102dd92

Mensaje cifrado en HEX = 69ac4ee7c4c552537a00a19bcaf7f0aaed7c9c8f769956a09bce6fadef6c3535f2211c9467067cf5c4a842ab

Mensaje cifrado en B64 = aaxO58TFUlN6AKGbyvfwqu18nI92mVagm85vre9sNTXyIRyUZwZ89cSoQqs=

Mensaje en claro = KeepCoding te enseña a codificar y a cifrar

Para mejorar el sistema deberíamos primero que el nonce no se repita por lo tanto habría que generar uno aleatorio y aparte utilizar el algoritmo chacha-20.poly ya que es una versión mejorada y utiliza código de autentificación por lo que es más seguro.

**Chacha20-poly**

Mensaje: {"nonce": "zJgnbXojBJIaMgdO", en b64 y generador aleatoriamente.

"datos asociados en base64 con esto no ciframos si no que autenticamos": "RGF0b3Mgbm8gY2lmcmFkb3Mgc8OzbG8gYXV0ZW50aWNhZG9z",

"texto cifrado en base64": "FwD6913lkNrtJBBG0XjOH1U5ElT7+WnYYm+Igw5+NmOQyLe8LujtXo4MuPI="

"tag": "P2wepn17sO3z0u1/lcJQ8w=="} código de autenticación

Datos cifrados en claro = KeepCoding te enseña a codificar y a cifrar.

Con el nonce y la clave es con lo que ciframos el mensaje el nonce debe ser único por mensaje. Por esta razón con Chacha20-poly mejoramos notablemente el sistema.

**4.** Tenemos el siguiente jwt, cuya clave es “Con KeepCoding aprendemos”.

eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c3VhcmlvIjoiRG9uIFBlcGl0byBkZSBsb3MgcGFsb3RlcyIsInJvbCI6ImlzTm9ybWFsIiwiaWF0IjoxNjY3OTMzNTMzfQ.gfhw0dDxp6oixMLXXRP97W4TDTrv0y7B5YjD0U8ixrE

¿Qué algoritmo de firma hemos realizado?

¿Cuál es el body del jwt?

Un hacker está enviando a nuestro sistema el siguiente jwt:

eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c3VhcmlvIjoiRG9uIFBlcGl0byBkZSBsb3MgcGFsb3RlcyIsInJvbCI6ImlzQWRtaW4iLCJpYXQiOjE2Njc5MzM1MzN9.krgBkzCBQ5WZ8JnZHuRvmnAZdg4ZMeRNv2CIAODlHRI

¿Qué está intentando realizar?

¿Qué ocurre si intentamos validarlo con pyjwt?

**Respuesta 1.**

El algoritmo que usamos es el HS256

**Respuesta 2.**

Body del jwt:

{

"usuario": "Don Pepito de los palotes",

"rol": "isNormal",

"iat": 1667933533

}

**Respuesta 3 hacker.**

El hacker está modificando el rol del body de “isNormal” a “IsAdmin”

**Respuesta 4.**

Si intentamos validarlo nos da un código de verificación distinto.

**5.** El siguiente hash se corresponde con un SHA3 Keccak del texto “En KeepCoding aprendemos cómo protegernos con criptografía”.

bced1be95fbd85d2ffcce9c85434d79aa26f24ce82fbd4439517ea3f072d56fe

¿Qué tipo de SHA3 hemos generado?

Y si hacemos un SHA2, y obtenemos el siguiente resultado:

4cec5a9f85dcc5c4c6ccb603d124cf1cdc6dfe836459551a1044f4f2908aa5d63739506f6468833d77c07cfd69c488823b8d858283f1d05877120e8c5351c833

¿Qué hash hemos realizado?

Genera ahora un SHA3 Keccak de 256 bits con el siguiente texto: “En KeepCoding aprendemos cómo protegernos con criptografía.” ¿Qué propiedad destacarías del hash, atendiendo a los resultados anteriores?

**Respuesta 1.**

Hemos generado un SHA3 256

**Respuesta 2.**

Hemos generado un SHA2 512

**Respuesta 3.**

302be507113222694d8c63f9813727a85fef61a152176ca90edf1cfb952b19bf

Vemos que la longitud del algoritmo es igual que el primero de SHA3 pero nos da distinto ya que la frase que hemos usado para calcular este último tiene un punto final y eso hace que varie. Respecto a SHA 2 vemos que varia mucho mas la longitud ya que es un código con más bytes.

**6.** Calcula el hmac-256 (usando la clave contenida en el Keystore) del siguiente texto:

Siempre existe más de una forma de hacerlo, y más de una solución válida.

Se debe evidenciar la respuesta. Cuidado si se usan herramientas fuera de los lenguajes de programación, por las codificaciones es mejor trabajar en hexadecimal.

**Clave:** A212A51C997E14B4DF08D55967641B0677CA31E049E672A4B06861AA4D5826EB

**Hmac-256:** 857d5ab916789620f35bcfe6a1a5f4ce98200180cc8549e6ec83f408e8ca0550

**7.** Trabajamos en una empresa de desarrollo que tiene una aplicación web, la cual requiere un login y trabajar con passwords. Nos preguntan qué mecanismo de almacenamiento de las mismas proponemos.

Tras realizar un análisis, el analista de seguridad propone un hash SHA-1. Su responsable, le indica que es una mala opción. ¿Por qué crees que es una mala opción?

Después de meditarlo, propone almacenarlo con un SHA-256, y su responsable le pregunta si no lo va a fortalecer de alguna forma. ¿Qué se te ocurre?

Parece que el responsable se ha quedado conforme, tras mejorar la propuesta del SHA-256, no obstante, hay margen de mejora. ¿Qué propondrías?

**Respuesta.**

SHA1, seria una mala opción ya que es un cifrado totalmente vulnerable y desde el año 2017 Google lo considera como un certificado inseguro. Uno de los fallos es que cuando el mismo valor de hash es producido para dos mensajes diferentes, puede ser explotado para falsificar firmas digitales y poder interceptar y descifrar comunicaciones cifradas con este cifrado.

Se podría mejora el SHA-256 integrando la implementación de autenticación de un JWT.

Ya que es JWT es un standard de transmisión de información en formato JSON de forma segura. Que se usa para implementar la autenticación de usuarios en aplicaciones. Se puede cifrar para ocultar su contenido.

El servidor de autenticación verifica los datos de login y si son correctos crea un

JWT para ese usuario (normalmente con una expiración).

También se podría mejorar mas utilizando el SHA-512 ya que es mas seguro aun porque produce expresiones de 64 bits de largo.

**8.** Como se puede ver en el API, tenemos ciertos parámetros que deben mantenerse confidenciales. Así mismo, nos gustaría que nadie nos modificase el mensaje sin que nos enterásemos. Se requiere una redefinición de dicha API para garantizar la integridad y la confidencialidad de los mensajes. Se debe asumir que el sistema end to end no usa TLS entre todos los puntos.

¿Qué algoritmos usarías?

**Respuesta.**

Usaríamos el AES-GCM. Ya que AES es el cifrado simétrico más seguro para proteger las comunicaciones y con GCM podemos dotar a AES tanto de confidencialidad como de autenticidad de los datos (integridad) en la misma operación. Así comprobamos que los datos transferidos no han sido modificados, es decir, autenticidad a los datos.

**9. Se requiere calcular el KCV de las siguientes claves AES:**

A2CFF885901A5449E9C448BA5B948A8C4EE377152B3F1ACFA0148FB3A426DB72

Para lo cual, vamos a requerir el KCV(SHA-256) así como el KCV(AES). El KCV(SHA-256) se corresponderá con los 3 primeros bytes del SHA-256. Mientras que el KCV(AES) se corresponderá con cifrar un texto del tamaño del bloque AES (16 bytes) compuesto con ceros binarios (00), así como un iv igualmente compuesto de ceros binarios. Obviamente, la clave usada será la que queremos obtener su valor de control.

**Respuestas.**

KCV (AES): 5244db

KCV SHA256: db7df2

**10.** El responsable de Raúl, Pedro, ha enviado este mensaje a RRHH:

“Se debe ascender inmediatamente a Raúl. Es necesario mejorarle sus condiciones económicas un 20% para que se quede con nosotros”.

Lo acompaña del siguiente fichero de firma PGP (MensajeRespoDeRaulARRHH.txt.sig).

Nosotros, que pertenecemos a RRHH vamos al directorio a recuperar la clave para verificarlo. Tendremos los ficheros Pedro-priv.txt y Pedro-publ.txt, con las claves privada y pública.

Las claves de los ficheros de RRHH son RRHH-priv.txt y RRHH-publ.txt que también se tendrán disponibles.

Se requiere verificar la misma, y evidenciar dicha prueba.

Así mismo, se requiere firmar el siguiente mensaje con la clave correspondiente de las anteriores, simulando que eres personal de RRHH.

Viendo su perfil en el mercado, hemos decidido ascenderle y mejorarle un 25% su salario. Saludos.

Por último, cifra el siguiente mensaje tanto con la clave pública de RRHH como la de Pedro y adjunta el fichero con la práctica.

Estamos todos de acuerdo, el ascenso será el mes que viene, agosto, si no hay sorpresas.

**Respuesta.**

**Pantallazos en el archivo de procesos.**

**11.** Nuestra compañía tiene un contrato con una empresa que nos da un servicio de almacenamiento de información de videollamadas. Para lo cual, la misma nos envía la clave simétrica de cada videollamada cifrada usando un RSA-OAEP. El hash que usa el algoritmo interno es un SHA-256.

El texto cifrado es el siguiente:



Las claves pública y privada las tenemos en los ficheros clave-rsa-oaep-publ.pem y clave-rsa-oaep-priv.pem.

Si has recuperado la clave, vuelve a cifrarla con el mismo algoritmo. ¿Por qué son diferentes los textos cifrados?

**Clave:**

e2cff885901a5449e9c448ba5b948a8c4ee377152b3f1acfa0148fb3a426db72

**Mensaje cifrado:**

.

**Respuesta.**

Los textos cifrados son distintos por el paddign del RSA-OAEP.

**12.** Nos debemos comunicar con una empresa, para lo cual, hemos decidido usar un algoritmo como el AES/GCM en la comunicación. Nuestro sistema, usa los siguientes datos en cada comunicación con el tercero:

Key:E2CFF885901B3449E9C448BA5B948A8C4EE322152B3F1ACFA0148FB3A426DB74

Nonce:9Yccn/f5nJJhAt2S

¿Qué estamos haciendo mal?

Cifra el siguiente texto:

He descubierto el error y no volveré a hacerlo mal

Usando para ello, la clave, y el nonce indicados. El texto cifrado preséntalo en hexadecimal y en base64.

**Respuesta.**

Lo que está mal es el nonce que siempre tiene que ser un valor aleatorio y jamás se puede repetir. Y la clave en el caso del AES también debe generarse de forma segura y con aleatoriedad, pero no tiene porque generarse en cada cifrado.

**Texto cifrado base 64:**

Xcu2Jh0PuinOOUMemgE7NMvKKk4Euy2QFJ1h9K/QTWXiq92dhLum64MHCV9QePv8FiVt3Q==

**Texto cifrado hexadecimal:**

5dcbb6261d0fba29ce39431e9a013b34cbca2a4e04bb2d90149d61f4afd04d65e2abdd9d84bba6eb8307095f5078fbfc16256ddd

**13.** Se desea calcular una firma con el algoritmo PKCS#1 v1.5 usando las claves contenidas en los ficheros clave-rsa-oaep-priv y clave-rsa-oaep-publ.pem del mensaje siguiente:

El equipo está preparado para seguir con el proceso, necesitaremos más recursos.

¿Cuál es el valor de la firma en hexadecimal?

Calcula la firma (en hexadecimal) con la curva elíptica ed25519, usando las claves ed25519-priv y ed25519-publ.

**Respuesta 1.**

Valor de la firma:



**Respuesta 2.**

b'bf32592dc235a26e31e231063a1984bb75ffd9dc5550cf30105911ca4560dab52abb40e4f7e2d3af828abac1467d95d668a80395e0a71c51798bd54469b7360d'

**14.** Necesitamos generar una nueva clave AES, usando para ello una HKDF (HMAC-based Extract-and-Expand key derivation function) con un hash SHA-512. La clave maestra requerida se encuentra en el keystore con la etiqueta “cifrado-sim-aes-256”. La clave obtenida dependerá de un identificador de dispositivo, en este caso tendrá el valor en hexadecimal:

e43bb4067cbcfab3bec54437b84bef4623e345682d89de9948fbb0afedc461a3

**clave maestra “cifrado-sim-aes-256”:**

A2CFF885901A5449E9C448BA5B948A8C4EE377152B3F1ACFA0148FB3A426DB72

**Respuesta:**

Clave Cifrado: e716754c67614c53bd9bab176022c952a08e56f07744d6c9edb8c934f52e448a

Clave MAC: e017b4756fce5d187a78fa0d1dfb42686991bd35098dd2b0c215b805e397f17c

**15. Nos envían un bloque TR31:**

D0144D0AB00S000042766B9265B2DF93AE6E29B58135B77A2F616C8D515ACDBE6A5626F79FA7B4071E9EE1423C6D7970FA2B965D18B23922B5B2E5657495E03CD857FD37018E111B

Donde la clave de transporte para desenvolver (unwrap) el bloque es:

A1A10101010101010101010101010102

**¿Con qué algoritmo se ha protegido el bloque de clave?**

Bloque de claves protegido mediante el método de vinculación de derivación de claves AES.

**¿Para qué algoritmo se ha definido la clave?**

AES.

**¿Para qué modo de uso se ha generado?**

Clave simétrica para cifrado de datos.

**¿Es exportable?**

Si, es exportable bajo clave no fiable.

**¿Para qué se puede usar la clave?**

Para encriptar y desencriptar.

**¿Qué valor tiene la clave?**

c1c1c1c1c1c1c1c1c1c1c1c1c1c1c1c1